BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平6－266562

(43)公開日　平成６年(1994)９月22日

(51)Int.Cl.[5]	識別記号	庁内整理番号	Ｆ Ｉ	技術表示箇所
Ｇ０６Ｆ　9/44	３３０　Ｚ	9193－5B		
9/45				
		9292－5B	Ｇ０６Ｆ　9/ 44　３２２　Ｆ	

審査請求　有　請求項の数３　ＦＤ　（全　８　頁）

(21)出願番号　特願平5－80164

(22)出願日　平成５年(1993)３月15日

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号

(72)発明者　宮本　敬士
東京都港区芝五丁目７番１号　日本電気株式会社内

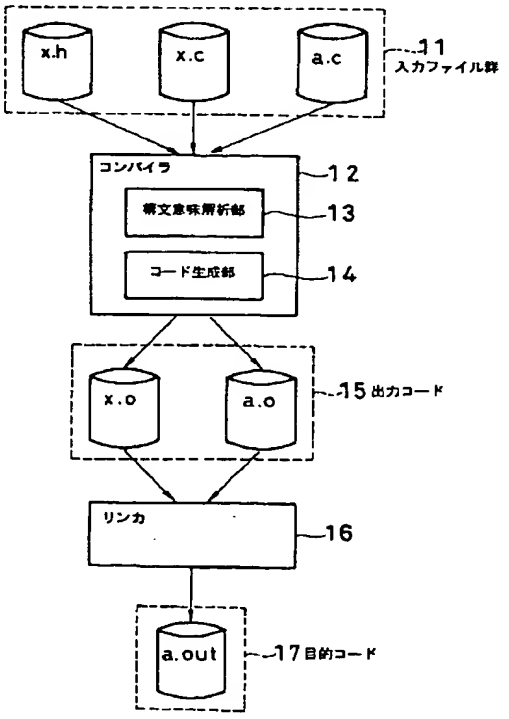
(74)代理人　弁理士　河原　純一

(54)【発明の名称】　オブジェクト指向言語処理システムにおける目的コードサイズ最適化方式

(57)【要約】

【目的】　目的コード内にメソッドアドレス格納テーブルの実体を１つのみ生成し、目的コードの大きさを削減する。

【構成】　入力ファイル群１１の各入力ファイルに対して出力コード１５を生成し、かつオブジェクト指向言語のメソッドの実行時決定を目的コード１７内に静的に生成されるメソッドアドレス格納テーブルを使用することによって実現するオブジェクト指向言語処理システムにおいて、構文意味解析部１３は、クラス宣言の意味解析処理時にメソッドの所属するクラス名と入力ファイル名とを比較し、一致すればメソッドアドレス格納テーブルの実体および宣言を出力コード１５内に生成し、一致しなければメソッドアドレス格納テーブルの宣言のみを出力コード１５内に生成する。リンカ１６は、複数の出力コード１５間の参照関係を解決して１つの目的コード１７を生成する。



【特許請求の範囲】

【請求項１】　複数の入力ファイルに対して各々に対応する出力コードを生成した後に複数の出力コード間の参照関係を解決して１つの目的コードを生成し、かつオブジェクト指向言語のメソッドの実行時決定を目的コード内に静的に生成されるメソッドアドレス格納テーブルを使用することによって実現するオブジェクト指向言語処理システムにおいて、

クラス宣言の意味解析処理時にメソッドの所属するクラス名と入力ファイル名とを比較し、一致すればメソッドアドレス格納テーブルの実体および宣言を出力コード内に生成し、一致しなければメソッドアドレス格納テーブルの宣言のみを出力コード内に生成する構文意味解析部を設けたことを特徴とするオブジェクト指向言語処理システムにおける目的コードサイズ最適化方式。

【請求項２】　前記クラス名と入力ファイル名とが一致しない場合は、オブジェクト指向言語処理システムが処理対象としているすべての入力ファイル名を調べ、入力ファイル名がクラス名と一致する入力ファイルがあればその入力ファイルの内容を走査し、現在対象としているクラスの宣言が含まれていなければその入力ファイル内にクラスの宣言を含め、入力ファイル名がクラス名と一致する入力ファイルがなければクラス名と同名の入力ファイルを作成し、この入力ファイルにクラスの宣言を含めるとともにこの入力ファイルを入力ファイル群に含める請求項１記載の目的コードサイズ最適化方式。

【請求項３】　前記オブジェクト指向言語がＣ＋＋でなり、前記メソッドアドレス格納テーブルが仮想関数テーブルでなる請求項１または２記載のオブジェクト指向言語処理システムにおける目的コードサイズ最適化方式。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明はオブジェクト指向言語（例えば、プログラミング言語Ｃ＋＋，ＬＯＯＰＳ，ＴＡＯ等）を用いた情報処理システムに関し、特に複数の入力ファイルに対して各々に対応する出力コードを生成した後に複数の出力コード間の参照関係を解決して１つの目的コードを生成し、かつオブジェクト指向言語のメソッドの実行時決定を目的コード内に静的に生成されるメソッドアドレス格納テーブルを使用することによって実現するオブジェクト指向言語処理システムにおける目的コードサイズ最適化方式に関する。

【０００２】

【従来の技術】オブジェクト指向言語にはメソッドの実行時決定という機能がある。これは異なるクラスに属するオブジェクトを統括して扱うために、オブジェクトに与えられたメッセージに対して起動されるメソッドをオブジェクトの所属するクラスに基づいて実行時に決定する機能である。

【０００３】ある種のオブジェクト指向言語処理システムでは、この機能を目的コード内に静的に生成されるメソッドアドレス格納テーブルを使用することによって実現している。実際に実行されるメソッドはオブジェクトの所属するクラスに基づいて決定されるから、メソッドアドレス格納テーブルの内容はクラスごとに（正確にはクラスとクラスの継承関係ごとに）異なるものである。

【０００４】従来は、実行時に決定されるメソッドの属するクラスの宣言が入力ソースプログラムファイル（以下、単に入力ファイルという）に含まれていると、オブジェクト指向言語処理システムのユーザが特に指定しない場合は出力コード内にメソッドアドレス格納テーブルの実体が必ず生成されていた。

【０００５】以下、オブジェクト指向言語の１つであるＣ＋＋の場合について従来技術を説明する。Ｃ＋＋には、仮想関数というメソッドの実行時決定のための機構がある。仮想関数であることを指定するためには、ｖｉｒｔｕａｌというキーワードを用いる。

【０００６】仮想関数を実現するためには、与えられたメッセージ呼出しに対して、実行時（プログラム動作時）に呼び出されるメソッドを決定する必要がある。仮想関数の呼出しが正しく行われるためには、実行されるメソッドが各オブジェクトのもつ情報をもとにプログラム動作時に決定されなければならない。

【０００７】これを実現する方法として自然に考えられる方法は、Ｃ＋＋処理系がクラスの属性を表す情報を出力コード中に埋め込み、各オブジェクトはその情報へのポインタを保持するという方法である。当然のことながら、クラスの属性を表す情報には実行されるメソッドの情報も含まれている。プログラム動作時には、オブジェクトからポインタをたどってこの情報をアクセスし、実行されるべきメソッドを決定することになる。

【０００８】仮想関数テーブルは、Ｃ＋＋処理系が仮想関数機能を実現するために出力コード中に埋め込むメソッドアドレス格納テーブルである。仮想関数テーブルは、「クラスの属性を表す情報」を簡略化したものと考えてよい。すなわち、クラスに関する全ての情報をもつのではなく、実行されるべきメソッドの決定を行うために必要最低限の情報をもっているテーブルである。ここでは、仮想関数テーブルは、実行されるべきメソッドのアドレス（位置）を情報としてもっていると考えてよい。

【０００９】仮想関数テーブルが必要なのは、仮想関数（ｖｉｒｔｕａｌ付きの関数）がクラス宣言中に用いられた場合である。仮想関数が書き換えられるごとに、言い換えれば仮想関数が書き換えられるようなクラスが宣言されるごとに仮想関数テーブルを生成しなければならない。

【００１０】また、入力ファイル中でオブジェクトを使用するには、そのオブジェクトが所属するクラスの宣言が入力ファイル中に含まれていなければならない。ある

クラスのオブジェクトを複数の入力ファイルで使用する場合、複数の入力ファイルにクラスの宣言が含まれていなければならない。このために、通常、クラスの宣言をヘッダファイル（ファイル名末尾を．ｈとすることが多い）に入れ、オブジェクトを使用する入力ファイルでヘッダファイルをインクルードする（Ｃ＋＋の言語仕様で、＃ｉｎｃｌｕｄｅを用いて指定する）ことが多い。ヘッダファイルのインクルードを指定すると、Ｃ＋＋処理系は、＃ｉｎｃｌｕｄｅが指定された部分をインクルードされるヘッダファイルの内容で置き換える。

【００１１】Ｃ＋＋処理系が出力コード中に仮想関数テーブルを埋め込むとき、仮想関数テーブルは、ソースプログラム中の外部変数と同じ形態で埋め込まれるが、外部変数と同じ形態で埋め込まれるためには、外部変数と同様に、仮想関数テーブルに名前（変数名と同様）が与えられる必要がある。このように名前が与えられるから、オブジェクトが仮想関数テーブルへのポインタを保持することが可能なのである（名前を使って参照することができる）。逆にいうと、仮想関数テーブルの名前さえわかっていれば、仮想関数テーブルの実体（先に述べた外部変数と同様な形態で出力コード中に確保される領域）は必要でない。

【００１２】ただし、重要なことは、単一または複数の入力ファイルで構成される全体のプログラムに対して生成される目的コード中に最低１個の仮想関数テーブルの実体が必要なことである。そうしないと、プログラムの実行中に、オブジェクトは実際に仮想関数テーブルが作成されていない位置のアドレスを保持することになってしまう。

【００１３】そこで、Ｃ＋＋処理系が全てのプログラムを同時に調べて目的コード中に仮想関数テーブルの実体を１個だけ生成するという処理を行えればよいが、しかし全てのプログラムを同時に調べるということは処理時間，消費メモリ量等の問題から現実的ではない。このため、多くのＣ＋＋処理系では、入力ファイルごとに出力コードを生成し、後で名前の結合（ある出力コードでは変数の名前だけを用いて参照し、別の出力コード中にその変数の領域があるという場合に、参照された部分の名前を実際の領域のアドレスに書き換える処理）を行うという方法をとっている。

【００１４】したがって、通常のＣ＋＋処理系では、各出力コードにおいてすべて仮想関数テーブルの実体を生成せざるを得ない。なぜならば、単一の入力ファイルを対象としている限り、他の入力ファイルに対する出力コードで仮想関数テーブルの実体が生成されているかどうかはわからないからである。

【００１５】

【発明が解決しようとする課題】上述した従来のオブジェクト指向言語処理システムでは、メソッドアドレス格納テーブルに関しては、実行時に決定されるメソッドの

属するクラスの宣言が入力ファイルに含まれていると、オブジェクト指向言語処理システムのユーザが特に指定しない場合は出力コード内にメソッドアドレス格納テーブルの実体が必ず生成されていたので、本来、目的コード内に実体がクラス毎に１つ生成されれば十分であるメソッドアドレス格納テーブルがクラスの宣言が取り込まれた入力ファイルの数だけ生成され、目的コードのサイズが不必要に大きくなるという問題点があった。

【００１６】この問題点に対処するために、オブジェクト指向言語処理システムのユーザが、入力ファイルごとに出力コード内にメソッドアドレス格納テーブルの実体および宣言を生成するか、あるいはメソッドアドレス格納テーブルの宣言のみを生成するかを指定し、複数の出力コード間の参照関係が解決されることにより目的コード内に生成されるメソッドアドレス格納テーブルの実体の数を減少させるという方法も存在する（ＡＴ＆Ｔ，“ＡＴ＆Ｔ　Ｃ＋＋　ＬＡＮＧＵＡＧＥ　ＳＹＳＴＥＭ　ＲＥＬＥＡＳＥ　２．１　Ｒｅｌｅａｓｅ　Ｎｏｔｅｓ”，ｐｐ．４－１６～４－１８）。しかし、クラスの宣言が入力ファイルに含まれているかどうかを、オブジェクト指向言語処理システムのユーザが判定するのは困難であるという問題点があった。

【００１７】目的コードのサイズを不必要に大きくしないためには、メソッドの実行時決定に用いられるメソッドアドレス格納テーブルの数を入力コードの意味が変わらない限り少なくすべきである。また、メソッドアドレス格納テーブルの数を減少させる処理は、オブジェクト指向言語処理システムのユーザの判断によることなく自動的に行うべきである。

【００１８】本発明の目的は、上述の点に鑑み、オブジェクト指向言語処理システムが出力コードへのメソッドアドレス格納テーブルの実体および／または宣言の生成を制御することにより、目的コード内にメソッドアドレス格納テーブルの実体を１つのみ生成し、目的コードの大きさを削減するようにしたオブジェクト指向言語処理システムにおける目的コードサイズ最適化方式を提供することにある。

【００１９】

【課題を解決するための手段】本発明のオブジェクト指向言語処理システムにおける目的コードサイズ最適化方式は、複数の入力ファイルに対して各々に対応する出力コードを生成した後に複数の出力コード間の参照関係を解決して１つの目的コードを生成し、かつオブジェクト指向言語のメソッドの実行時決定を目的コード内に静的に生成されるメソッドアドレス格納テーブルを使用することによって実現するオブジェクト指向言語処理システムにおいて、クラス宣言の意味解析処理時にメソッドの所属するクラス名と入力ファイル名とを比較し、一致すればメソッドアドレス格納テーブルの実体および宣言を出力コード内に生成し、一致しなければメソッドアドレ

ス格納テーブルの宣言のみを出力コード内に生成する構文意味解析部を設けたことを特徴とする。

【００２０】

【実施例】次に、本発明について図面を参照して詳細に説明する。

【００２１】本発明の一実施例として、オブジェクト指向言語で記述されたソースプログラムを入力とし、構文意味解析部においてこれを中間言語形式に変換し、さらにコード生成部でこれを出力コードに変換してファイルに出力する、コンパイル方式のオブジェクト指向言語処理システムの例を挙げる。

【００２２】図１は、本発明の一実施例に係るオブジェクト指向言語処理システムにおける目的コードサイズ最適化方式の構成を示すブロック図である。本実施例のオブジェクト指向言語処理システムにおける目的コードサイズ最適化方式は、入力ファイル群１１と、構文意味解析部１３およびコード生成部１４を含むコンパイラ１２と、出力コード１５と、リンカ１６と、目的コード１７とから構成されている。

【００２３】図２を参照すると、構文意味解析部１３の目的コードサイズ最適化処理は、クラス宣言意味解析処理ステップＳ１と、メソッドアドレス格納テーブル要否判定ステップＳ２と、入力ファイル名／クラス名比較ステップＳ３と、メソッドアドレス格納テーブル実体および宣言生成ステップＳ４と、メソッドアドレス格納テーブル宣言生成ステップＳ５と、同名入力ファイル存在判定ステップＳ６と、クラス宣言存在判定ステップＳ７と、クラス宣言包含ステップＳ８と、コンパイル済判定ステップＳ９と、コンパイル未処理設定ステップＳ１０と、同名入力ファイル作成ステップＳ１１と、クラス宣言包含ステップＳ１２と、コンパイル未処理設定ステップＳ１３とからなる。

【００２４】次に、このように構成された本実施例のオブジェクト指向言語処理システムにおける目的コードサイズ最適化方式の動作について説明する。

【００２５】コンパイラ１２は、入力ファイル群１１として複数の入力ファイルを受け付ける。コンパイラ１２は、入力ファイル群１１を構成するすべての入力ファイルについて、１つずつコンパイル処理を行うために構文意味解析部１３およびコード生成部１４を起動する。

【００２６】コンパイラ１２において、目的コードサイズ最適化処理は、構文意味解析部１３によって行われる。

【００２７】最初に、構文意味解析部１３は、入力ファイル内のクラス宣言の意味解析処理を行う（ステップＳ１）。詳しくは、クラス宣言のメンバのシンボルテーブルへの登録等を行う。

【００２８】次に、構文意味解析部１３は、入力ファイル内のクラス宣言がメソッドアドレス格納テーブルを必要とするかどうかを調べる（ステップＳ２）。これは、

入力ファイルに記述されているプログラミング言語の仕様に基づき、現在処理対象としているクラスがメソッドの実行時決定を必要とするかどうかにより決定される。Ｃ＋＋処理系の場合、ｖｉｒｔｕａｌの宣言があるか（＃ｉｎｃｌｕｄｅされる場合を含める）どうかに基づいて決定する。メソッドアドレス格納テーブルを必要としないならば、このクラスに対してこれ以上の処理は行わない。

【００２９】現在処理対象としているクラスがメソッドアドレス格納テーブルを必要とする場合、構文意味解析部１３は、クラス名と入力ファイル名とが等しいかどうかを調べる（ステップＳ３）。ただし、入力ファイル名にオブジェクト指向言語処理システム自体またはオブジェクト指向言語処理システムがその上で動作する情報処理システムが定めた付加部分（拡張子）が含まれている場合は、その付加部分を除いた部分をクラス名と比較する。以後、クラス名と入力ファイル名とを比較するときはこの処理を行うものとする。

【００３０】クラス名と入力ファイル名とが等しいならば、構文意味解析部１３は、出力コード内にメソッドアドレス格納テーブルの実体および宣言を生成して（ステップＳ４）、このクラスに対してこれ以上の処理は行わない。この結果、メソッドアドレス格納テーブルの実体および宣言は、クラス名と入力ファイル名とが一致する場合のみ出力コード内に生成されることになる。Ｃ＋＋処理系の場合、出力コード１５中のデータセクションに仮想関数テーブルを生成するとともに、シンボルテーブルにクラスの仮想関数テーブルの属性を生成する（図３参照）。

【００３１】クラス名と入力ファイル名とが等しくない場合、構文意味解析部１３は、出力コード内にメソッドアドレス格納テーブルの宣言のみを生成して（ステップＳ５）、クラス名と同名の入力ファイルがコンパイラ１２に与えられた入力ファイル群１１中に存在するかどうかを調べる（ステップＳ６）。

【００３２】入力ファイル群１１中にクラス名と同名の入力ファイルが存在する場合は、構文意味解析部１３は、クラス名と同名の入力ファイルの内容を調べ、現在対象としているクラスの宣言が含まれていれるかどうかを調べる（ステップＳ７）。クラスの宣言が含まれていれば、このクラスに対してこれ以上の処理を行わない。

【００３３】クラスの宣言が含まれていなければ、構文意味解析部１３は、クラス名と同名の入力ファイルの内容を変更して現在対象としているクラスの宣言を含める（ステップＳ８）。Ｃ＋＋処理系の場合、シンボルテーブルにクラスの仮想関数テーブルの属性を生成する（図３参照）。さらに、すでにコンパイル処理が終了している場合には（ステップＳ９でイエス）、コンパイラ１２が記録しているコンパイル未処理の入力ファイル群１１に含める（ステップＳ１０）。

【００３４】ステップＳ６で、クラス名と同名の入力ファイルがコンパイラ１２に与えられた入力ファイル群１１の中に存在しない場合は、構文意味解析部１３は、新たにクラス名と同名の入力ファイルを作成して（ステップＳ１１）、オブジェクト指向言語処理システムで動作している情報処理システムの適当な位置に置き、この入力ファイル内にクラスの宣言を含め（ステップＳ１２）、さらにこの入力ファイルをコンパイラ１２が記録しているコンパイル未処理の入力ファイル群１１に含める（ステップＳ１３）。この場合、新たに作成した入力ファイルについてコンパイル処理が施される時点で、出力コード１５中にメソッドアドレス格納テーブルの実体および宣言が生成される。

【００３５】コンパイラ１２により複数の出力コード１５が出力された後、リンカ１６は、複数の出力コード１５間の参照関係を解決して１つの目的コード１７を生成する。

【００３６】図３は、入力ファイルＸ．ｈ，Ｘ．Ｃおよびａ．Ｃと出力コードＸ．ｏおよびａ．ｏの一例を示す図である。入力ファイルＸ．Ｃおよびａ．Ｃにおいてヘッダファイル Ｘ．ｈがインクルードされている。クラス名Ｘと入力ファイル名Ｘ．Ｃの付加部分を除いたものとが一致するので、出力コードＸ．ｏにはクラスＸの仮想関数テーブルの実体および宣言が生成されている。また、クラス名Ｘと入力ファイル名ａ．Ｃの付加部分を除いたものとが一致しないので、出力コードａ．ｏにはクラスＸの仮想関数テーブルの宣言のみが生成されている。

【００３７】

【発明の効果】以上説明したように本発明は、オブジェクト指向言語処理システムの構文意味解析部に出力コード内へのメソッドアドレス格納テーブルの実体および／または宣言の生成の選択をオブジェクト指向言語処理システム自体が行う機能を設け、目的コード内にメソッドアドレス格納テーブルの実体を１つのみ生成するようにしたことにより、オブジェクト指向言語処理システムのユーザが困難な解析を行うことなく、目的コードのサイズを削減することができるという効果がある。

【図面の簡単な説明】

【図１】本発明の一実施例に係るオブジェクト指向言語処理システムにおける目的コードサイズ最適化方式の構成を示すブロック図である。

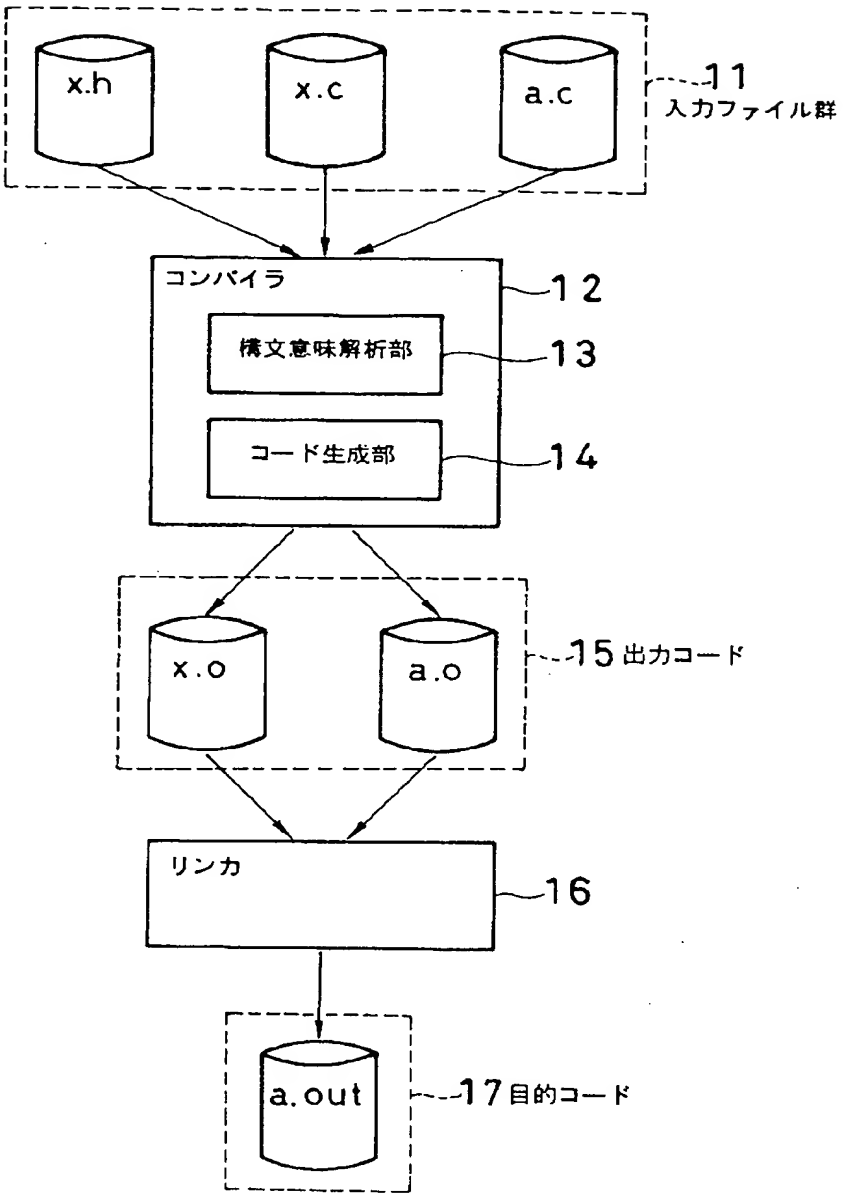
【図２】図１中の構文意味解析部における目的コードサイズ最適化処理を示す流れ図である。

【図３】図１中の入力ファイルおよび出力コードの一例を示す図である。

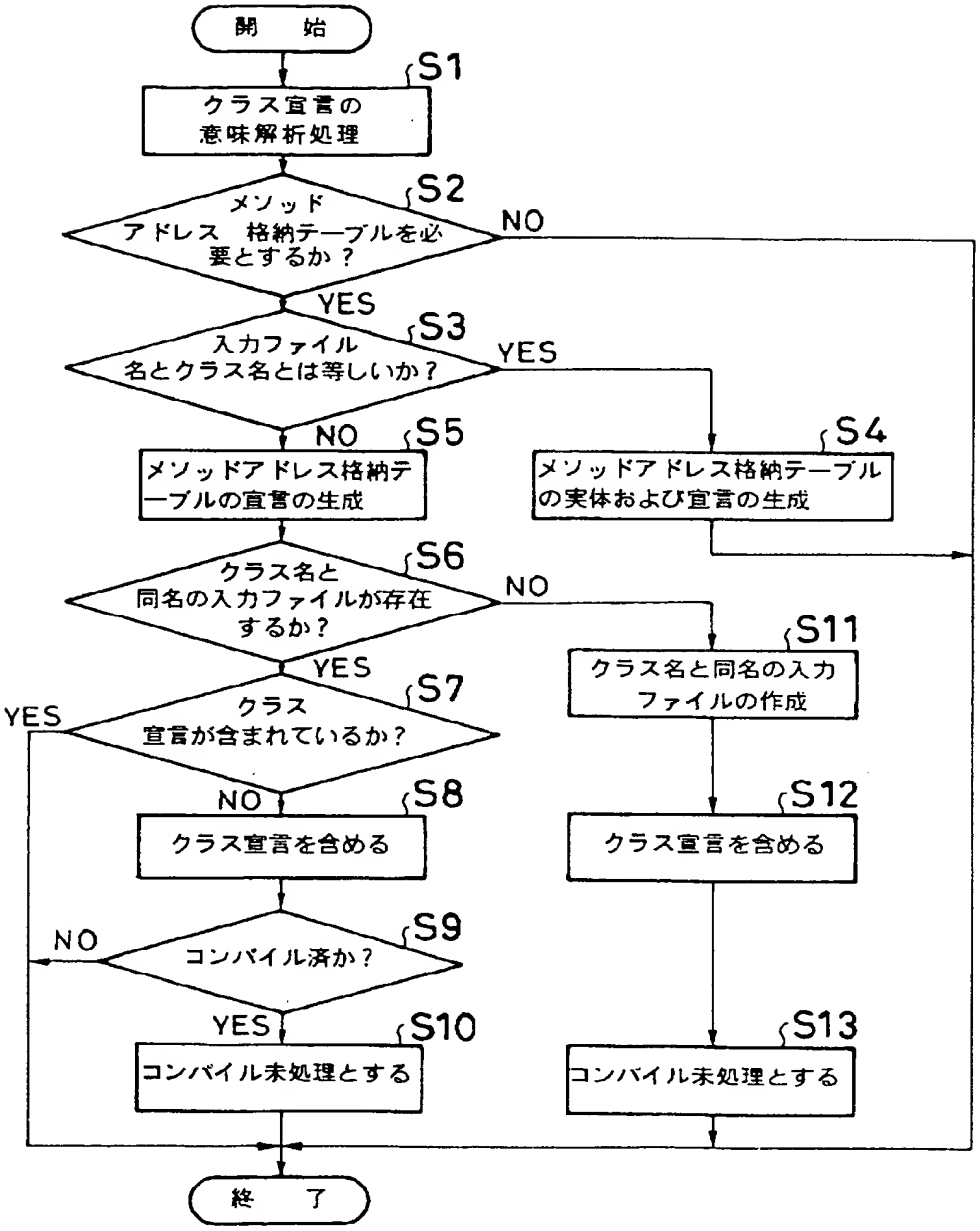
【符号の説明】

１１　入力ファイル群
１２　コンパイラ
１３　構文意味解析部
１４　コード生成部
１５　出力コード
１６　リンカ
１７　目的コード

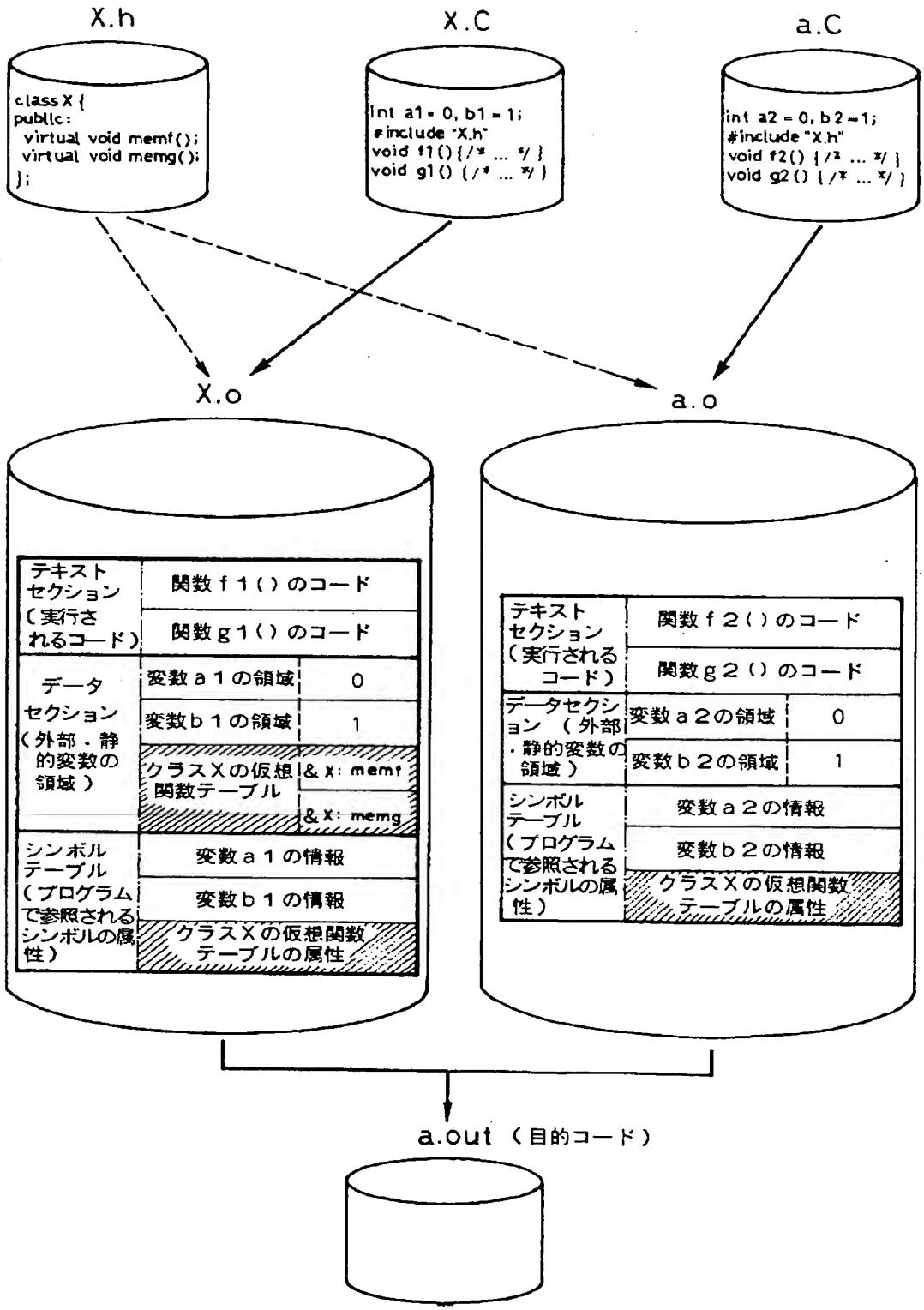
【図１】


x.h
x.c
a.c
11
入力ファイル群
コンパイラ
12
構文意味解析部
13
コード生成部
14
x.o
a.o
15 出力コード
リンカ
16
a.out
17 目的コード

【図２】


開　始
S1
クラス宣言の
意味解析処理
S2
メソッド
アドレス　格納テーブルを必
要とするか？
NO
YES
S3
入力ファイル
名とクラス名とは等しいか？
YES
NO
S5
メソッドアドレス格納テ
ーブルの宣言の生成
S4
メソッドアドレス格納テーブル
の実体および宣言の生成
S6
クラス名と
同名の入力ファイルが存在
するか？
NO
YES
S11
クラス名と同名の入力
ファイルの作成
S7
クラス
宣言が含まれているか？
YES
NO
S8
クラス宣言を含める
S12
クラス宣言を含める
S9
コンパイル済か？
NO
YES
S10
コンパイル未処理とする
S13
コンパイル未処理とする
終　了

[図３]


X.h
class X {
public:
virtual void memf();
virtual void memg();
};
X.C
int a1 = 0, b1 = 1;
#include "X.h"
void f1() { /* ... */ }
void g1() { /* ... */ }
a.C
int a2 = 0, b2 = 1;
#include "X.h"
void f2() { /* ... */ }
void g2() { /* ... */ }
X.o
テキストセクション（実行されるコード）
関数ｆ１（）のコード
関数ｇ１（）のコード
データセクション（外部・静的変数の領域）
変数ａ１の領域
0
変数ｂ１の領域
1
クラスＸの仮想関数テーブル
&X: memf
&X: memg
シンボルテーブル（プログラムで参照されるシンボルの属性）
変数ａ１の情報
変数ｂ１の情報
クラスＸの仮想関数テーブルの属性
a.o
テキストセクション（実行されるコード）
関数ｆ２（）のコード
関数ｇ２（）のコード
データセクション（外部・静的変数の領域）
変数ａ２の領域
0
変数ｂ２の領域
1
シンボルテーブル（プログラムで参照されるシンボルの属性）
変数ａ２の情報
変数ｂ２の情報
クラスＸの仮想関数テーブルの属性
a.out（目的コード）